Tidyverse II: Tidyr and Advanced Dplyr

Statistical Computing

Last week: Pipes and dplyr

- The tidyverse is a collection of packages for common data science tasks
- Tidyverse functionality is greatly enhanced using pipes (`%>%` operator)
- Pipes allow you to string together commands to get a flow of results
- `dplyr` is a package for data wrangling, with several key verbs (functions)
- `slice()` and `filter()`: subset rows based on numbers or conditions
- `select()`: select columns
- `arrange()`: order rows by one or multiple columns
- `rename()` and `mutate()`: rename or create columns
- `mutate_at()`: apply a function to given columns

Recall: using pipes (%>%)

Tidyverse functions are at their best when composed together using the pipe (`%>%`) operator!

```
library(tidyverse)
mtcars %>%
  filter(., (mpg >= 14 & disp >= 200) | (drat <= 3)) %>%
  head(., 2)
```

```
##    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## 1 21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## 2 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
```

Pro tip: use `ctrl + shift + m` in RStudio as a shortcut for typing `%>%`

Part I

Mastering the `tidyr` verbs

`tidyr` verbs

Our `tidyr` journey starts of with learning the following verbs (functions):

- `gather()`: make "wide" data longer
- `spread()`: make "long" data wider
- `separate()`: split a single column into multiple columns
- `unite()`: combine multiple columns into a single column

Key takeaway: as with `dplyr`, think of data frames as nouns and `tidyr` verbs as actions that you apply to manipulate them—especially natural when using pipes

gather()

Use `gather()` to make "wide" data longer:

```
library(devtools)
```

```
## Loading required package: usethis
```

```
devtools::install_github("rstudio/EDAWR")
```

```
## Skipping install of 'EDAWR' from a github remote, the SHA1 (fbfee984) has not changed since last ins
##   Use `force = TRUE` to force installation
```

```
library(EDAWR) # Load some nice data sets
```

```
##
## Attaching package: 'EDAWR'
```

```
## The following object is masked from 'package:dplyr':
##
##     storms
```

```
## The following objects are masked from 'package:tidyr':
##
##     population, who
```

```
#cases: TB cases in USA, Germany, and France

EDAWR::cases %>%
  head(., 3)
```

```
##   country  2011  2012  2013
## 1      FR  7000  6900  7000
## 2      DE  5800  6000  6200
## 3      US 15000 14000 13000
```

```
EDAWR::cases %>%
  gather(., "year", "n", 2:4) %>%
  head(., 5)
```

```
##   country year     n
## 1      FR 2011  7000
## 2      DE 2011  5800
## 3      US 2011 15000
## 4      FR 2012  6900
## 5      DE 2012  6000
```

- Here we transposed columns 2:4 into a `year` column
- We put the corresponding count values into a column called `n`
- Note `tidyr` did all the heavy lifting of the transposing work
- We just had to declaratively specify the output

spread()

Use `spread()` to make "long" data wider:

```
EDAWR::pollution %>%
  head(., 5)
```

```
##       city  size amount
## 1 New York large     23
## 2 New York small     14
## 3   London large     22
## 4   London small     16
## 5  Beijing large    121
```

```
EDAWR::pollution %>%
  spread(., size, amount)
```

```
##       city large small
## 1  Beijing   121    56
## 2   London    22    16
## 3 New York    23    14
```

```
# size: Size of air particulate measured. Fine suspended particles smaller than 10 microns
#in diameter (large) and 2.5 microns in diameter (small).
# amount: The mean annual concentration of particles in milligrams per meter cubed (ug/m3)
```

- Here we transposed to a wide format by size
- We tabulated the corresponding amount for each size
- Note `tidyr` did all the heavy lifting of
- We just had to declaratively specify the output
- Note that `spread()` and `gather()` are inverses

separate()

```
EDAWR::storms %>%
  head(., 3)
```

```
##     storm wind pressure       date
## 1 Alberto  110     1007 2000-08-03
## 2    Alex   45     1009 1998-07-27
## 3 Allison   65     1005 1995-06-03
```

```
storms2 <- EDAWR::storms %>%
  separate(., date, into = c("y", "m", "d"), sep="\\-")
storms2
```

```
## # A tibble: 6 x 6
##   storm    wind pressure y     m     d
##   <chr>   <int>    <int> <chr> <chr> <chr>
## 1 Alberto   110     1007 2000  08    03
## 2 Alex       45     1009 1998  07    27
## 3 Allison    65     1005 1995  06    03
## 4 Ana        40     1013 1997  06    30
## 5 Arlene     50     1010 1999  06    11
## 6 Arthur     45     1010 1996  06    17
```

```
df <- data.frame(x = c(NA, "a.b", "a.d", "b.c"))
df %>% separate(x, into = c("A", "B"),sep="\\.")
```

```
##      A    B
## 1 <NA> <NA>
## 2    a    b
## 3    a    d
## 4    b    c
```

```
df <- data.frame(x = c(NA, "a.b", "a.d", "b.c"))
df %>% separate(x, into = c("A", "B"), sep=1)
```

```
##      A    B
## 1 <NA> <NA>
## 2    a   .b
## 3    a   .d
## 4    b   .c
```

```
df <- data.frame(x = c(NA, "a?b", "a.d", "b:c"))
df %>% separate(x, into = c("A","B"), sep = "([\\.\\?\\:])")
```

```
##      A    B
## 1 <NA> <NA>
## 2    a    b
## 3    a    d
## 4    b    c
```

unite()

Use `unite()` to combine multiple columns into a single column:

```
storms2 %>%
  unite(., date, y, m, d, sep = "-")
```

```
## # A tibble: 6 x 4
##   storm    wind pressure date
##   <chr>   <int>    <int> <chr>
## 1 Alberto   110     1007 2000-08-03
## 2 Alex       45     1009 1998-07-27
```

```
## 3 Allison    65     1005 1995-06-03
## 4 Ana        40     1013 1997-06-30
## 5 Arlene     50     1010 1999-06-11
## 6 Arthur     45     1010 1996-06-17
```

Note that `unite()` and `separate()` are inverse operations

Part II

dplyr: group_by() and summarize()

Recall: dplyr and SQL

- Once you learn `dplyr` you should find SQL very natural, and vice versa!
- This will make it much easier for tasks that require using both R and SQL to munge data and build statistical models
- One major link is through powerful verbs like `group_by()` and `summarize()`, which are used to aggregate data (now)
- Another major link to SQL is through merging/joining data frames, via `left_join()` and `inner_join()` verbs (shortly)

group_by()

Use `group_by()` to define a grouping of rows based on a column:

```
mtcars %>%
  group_by(., cyl) %>%
  head(., 6)
```

```
## # A tibble: 6 x 11
## # Groups:   cyl [3]
##     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1  21       6   160   110  3.9   2.62  16.5     0     1     4     4
## 2  21       6   160   110  3.9   2.88  17.0     0     1     4     4
## 3  22.8     4   108    93  3.85  2.32  18.6     1     1     4     1
## 4  21.4     6   258   110  3.08  3.22  19.4     1     0     3     1
## 5  18.7     8   360   175  3.15  3.44  17.0     0     0     3     2
## 6  18.1     6   225   105  2.76  3.46  20.2     1     0     3     1
```

- This doesn't actually change anything about the way the df looks
- Only difference is that when it prints, we're told about the groups
- But it will play a big role in how other `dplyr` verbs act

summarise()

Use `summarise()` (or `summarize()` for us Americans) to apply functions to rows—ungrouped or grouped—of a data frame:

```
# Ungrouped
mtcars %>%
  summarise(.,
            mpg=mean(mpg),
            hp=mean(hp))
```

```
##        mpg       hp
## 1 20.09062 146.6875
```

```
# Grouped by number of cylinders
mtcars %>%
  group_by(., cyl) %>%
  summarise(.,
            mpg=mean(mpg),
            hp=mean(hp))
```

```
## # A tibble: 3 x 3
##     cyl   mpg    hp
##   <dbl> <dbl> <dbl>
## 1     4  26.7  82.6
## 2     6  19.7 122.
## 3     8  15.1 209.
```

```
mtcars %>%
  group_by(., cyl) %>%
  summarise(.,
            mpg_mean=mean(mpg),
            mpg_max=max(mpg),
            hp_mean=mean(hp),
            hp_max=max(hp))
```

```
## # A tibble: 3 x 5
##     cyl mpg_mean mpg_max hp_mean hp_max
##   <dbl>    <dbl>   <dbl>   <dbl>  <dbl>
## 1     4     26.7    33.9    82.6    113
## 2     6     19.7    21.4   122.     175
## 3     8     15.1    19.2   209.     335
```

ungroup()

Use `ungroup()` to remove groupings from a data frame:

```
mtcars %>%
  group_by(., cyl) %>%
  ungroup(.) %>%
  summarise(.,
            hp=mean(hp),
            mpg=mean(mpg))
```

```
## # A tibble: 1 x 2
##      hp   mpg
##   <dbl> <dbl>
## 1  147.  20.1
```

Simplifying indexing

```
str.url <- "http://www.stat.cmu.edu/~ryantibs/statcomp/data/trump.txt"
str.url %>%
  readLines %>%
  paste(collapse=" ") %>%
  strsplit(split="[[:space:]]|[[:punct:]]") %>%
  unlist(x = .) %>%
  .[. != ""] %>%
  table(.) %>%
  sort(decreasing=TRUE) %>%
  head(.)
```

```
## .
##  the  and   of   to  our will
##  189  146  127  126   90   83
```

This required a tricky indexing operation using `%>%` on vectors: `.[. != ""]`

We can perform the same operations using `dplyr` if we convert to a data frame:

```
str.url %>%
  readLines(.) %>%
  paste(collapse=" ") %>%
  strsplit(split="[[:space:]]|[[:punct:]]") %>%
  unlist(.) %>%
  as_data_frame() %>%            # Convert to a data frame (dplyr version)
  rename(word_list = value) %>%  # Rename the column to `word_list`
  filter(word_list != "") %>%    # Now indexing is easy!
  group_by(word_list) %>%
  summarise(word_count = n()) %>%
  arrange(desc(word_count)) %>%
  slice(1:6)
```

```
## Warning: `as_data_frame()` is deprecated, use `as_tibble()` (but mind the new semantics).
## This warning is displayed once per session.
```

```
## # A tibble: 6 x 2
##   word_list word_count
##   <chr>          <int>
## 1 the              189
## 2 and              146
## 3 of               127
## 4 to               126
## 5 our               90
## 6 will              83
```

Part III

dplyr: left_join() and inner_join()

Join operations

A "join" operation in database terminology is a merging of two data frames for us. There are 4 types of joins:

- **Inner join** (or just **join**): retain just the rows each table that match the condition
- **Left outer join** (or just **left join**): retain all rows in the first table, and just the rows in the second table that match the condition
- **Right outer join** (or just **right join**): retain just the rows in the first table that match the condition, and all rows in the second table
- **Full outer join** (or just **full join**): retain all rows in both tables

Column values that cannot be filled in are assigned `NA` values

Two toy data frames

```
tab1 = data.frame(name = c("Alexis", "Bernie", "Charlie"),
                  children = 1:3,
                  stringsAsFactors = FALSE)
tab2 = data.frame(name = c("Alexis", "Bernie", "David"),
                  age = c(54, 34, 63),
                  stringsAsFactors = FALSE)
tab1
```

```
##      name children
## 1  Alexis        1
## 2  Bernie        2
## 3 Charlie        3
```

```
tab2
```

```
##     name age
## 1 Alexis  54
## 2 Bernie  34
## 3  David  63
```

inner_join()

Suppose we want to join `tab1` and `tab2` by name, but keep only customers in intersection:

```
inner_join(x=tab1, y=tab2, by="name")
```

```
##     name children age
## 1 Alexis        1  54
## 2 Bernie        2  34
```

left_join()

Suppose we want to join `tab1` and `tab2` by name, but keep all customers from `tab1`:

```
left_join(x=tab1, y=tab2, by="name")
```

```
##      name children age
## 1  Alexis        1  54
## 2  Bernie        2  34
## 3 Charlie        3  NA
```

right_join()

Suppose we want to join `tab1` and `tab2` by name, but keep all customers from `tab2`:

```
right_join(x=tab1, y=tab2, by="name")
```

```
##     name children age
## 1 Alexis        1  54
## 2 Bernie        2  34
## 3  David       NA  63
```

full_join()

Finally, suppose we want to join `tab1` and `tab2` by name, and keep all customers from both:

```
full_join(x=tab1, y=tab2, by="name")
```

```
##      name children age
## 1  Alexis        1  54
## 2  Bernie        2  34
## 3 Charlie        3  NA
## 4   David       NA  63
```

Summary

- `tidyr` is a package for manipulating the structure of data frames
- `gather()`: make wide data longer
- `spread()`: make long data wider
- `unite()` and `separate()`: combine or split columns
- `dplyr` has advanced functionality that mirrors SQL
- `group_by()`: create groups of rows according to a condition
- `summarise()`: apply computations across groups of rows
- `*_join()` where `*` = `inner`, `left`, `right`, or `full`: join two data frames together according to common values in certain columns, and `*` indicates how many rows to keep

References

- tidyverse.org: the main place for package updates and news
- tidyverse forum: the official forum for questions
- dplyr+tidyr cheatsheet (2015 version) from RStudio
- dplyr cheatsheet (current version) from RStudio
- tidyr cheatsheet (current version) from RStudio